A Novel Reduction from #SAT to #2SAT Based on Symmetry: *Simply Drop the Large Clauses**

Bannach, Max[†] max.bannach@esa.int
Erik D. Demaine[‡] edemaine@mit.edu
Timothy Gomez[†] tagomez7@mit.edu
Markus Hecher[§] hecher@cril.fr

Abstract. The counting problem #2SAT is complete for #P under *Turing (many-call) reductions*, which dates back to the seminal work by Valiant from 1979. Arguably, this reduction is the *opposite* from being simple as it is a sophisticated chain of transformation from #SAT, via several variations of the problem of computing the permanent, to the task of counting matchings in graphs, and then finally to #2SAT. In contrast, we give *a simple reduction* that makes only two calls instead of polynomially many calls. Our reduction is based on the inclusion-exclusion principle from #SAT to *weighted* #2SAT with weights in $\{-1, 1\}$. From there, we map the computation of such weighted model counting problems to the subtraction of two unweighted model counting problems using parity constraints. We can encode these parity constraints using almost solely binary clauses and *very few clauses of size four*. Then, thanks to the subtraction and the symmetry between the two formulas, these clauses of size four can *simply be omitted* without changing the overall result of the computation — thus leading to a surprisingly simple reduction from #SAT to two calls of #2SAT.

Apart from the fact that the reduction is simple, it also improves the construction by Valiant in various ways. First, it is computable either in logspace or linear time and, hence, also produces formulas that are linearly bounded by the input formula (whereas the construction of Valiant produces formulas of polynomial size). Furthermore, the construction can easily be adapted to preserve structural parameters like the input's treewidth, yielding a simple proof of the fact that #SAT can be solved in time $2^k \operatorname{poly}(|\varphi|)$ on formulas of *incidence* treewidth k (the current proof of this fact requires an involved algorithm based on Möbius and Zeta transformations). Finally, our technique can be used to improve and simplify in the sparsification lemma from #dSAT to #2SAT (current proofs are based on block interpolation and are quite involved).

1 Introduction The complexity class #P is the canonical class in the realm of counting problems that contains many natural problems like #SAT[1]. In contrast to classical complexity classes like NP that appear to be robust, #P is relatively fragile in the following sense: Many problems that are complete for #P are *only* complete for the class under *Turing (many-call) reductions,* but are *not* under many-one reductions. An example is the problem #DNF, which asks to count the number of satisfying assignments of a propositional formula in disjunctive normal form. It is trivially in #P and can be shown to be complete by the following Turing reduction from #SAT: Given any CNF φ, its negation $\neg\varphi$ is a DNF and, if φ has n variables, we clearly have:

$$\#(\varphi) = 2^n - \#(\neg\varphi).$$

Note how we have to perform a subtraction *after* the evaluation of $\#(\neg\varphi)$ (the #DNF call) and, hence, it is indeed not a many-one reduction. We might be tempted to think that such a little subtraction is not a big deal, but it turns out that #DNF lies in a complexity class called spanL [1], which is a strict subset of #P unless NP = RP, as already #SAT is expected to be hard to approximate [30, 11], which is in contrast to spanL [2]. Therefore, there

*Authors are ordered alphabetically. The full version of the paper can be accessed at https://arxiv.org/abs/2506.06716. We thank all anonymous reviewers for their feedback. Research was partly funded by the Austrian Science Fund (FWF), grant J4656. Part of the research was carried out while Hecher was a postdoc at MIT and while he was at UC Berkeley while visiting the Simons institute for the theory of computing (part of the program *Logic and Algorithms in Database Theory and AI*).

[†]European Space Agency, AI and Data Science Section, Noordwijk, The Netherlands.
[‡]Massachusetts Institute of Technology, Computer Science and Artificial Intelligence Lab, USA.
[§]Univ. Artois, CNRS, UMR 8188, Centre de Recherche en Informatique de Lens (CRIL), France.

[1]The problem #SAT asks to count the number of satisfying assignments of a propositional formula.

is *no* many-one reduction from #SAT to #DNF unless NP = RP; but it is enough to perform arithmetic, e.g., a small subtraction in a *post-processing step* on top of the obtained count.

Another problem that is famously complete for #P under Turing reductions is #2SAT, the problem of counting the models of a CNF φ in which every clause contains at most two variables. In his seminal work [28, 29], Valiant proved that #2SAT is complete for #P by reducing #SAT to the *computation of the permanent of an integer matrix*, which he reduced to the *computation of the permanent of a binary matrix*, which can be reduced to *counting the number of perfect matchings in bipartite graphs*, which in turn can be reduced to *counting all matchings in bipartite graphs*, which finally reduces to #2SAT. As for #DNF, this chain of reductions must be a Turing reduction unless NL = NP and, indeed, many steps in Valiant's reduction are Turing reductions: Almost each step in the chain produced a *polynomial number* of instances of the next step, and the results of all of these instances need to be connected in a non-trivial manner using division and modulo operations. From a complexity-theoretic perspective, this may be concerning as we saw that already a little subtraction in the post-processing step can change the complexity of the problem — we may wonder how much of the complexity of #2SAT is obscured by this reduction?

This reduction is not only relevant from a complexity-theoretic perspective, but becomes increasingly important in practical applications. Model counting recently proved very useful in the verification and simulation of quantum circuits [22, 23] and practical tools are actively developed [12]. To that end, Valiant's reduction was, for instance, rephrased in graphical languages like the ZH-calculus leading to small improvements [17]. But also from a classroom perspective we can argue that Valiant's reduction is not optimal, as it easily fills a whole lecture without providing a lot of insights into the aspects that actually make #2SAT hard.

1.1 Contribution: A Simple Reduction from #SAT to #2SAT Our main contribution is an elementary simpler reduction from #SAT to #2SAT based on the inclusion-exclusion principle, whose main ingredient is given in Section 3. The correctness of the reduction is intuitively easy to grasp, making it well suited to be presented within a lecture about the topic. But the reduction also significantly improves the reduction by Valiant in various directions: Given a formula φ, we produce only two formulas ψ_1 and ψ_2 such that:

$$\#(\varphi) = \#(\psi_1) - \#(\psi_2).$$

There is no polynomial number of instances produced, and the results of the two #2SAT calls can be combined by a simple minus. Furthermore, ψ_1 and ψ_2 are easy to compute (in complexity-theoretic language in logspace, but conceptually it is also an easy construction); and both formulas are almost identical (they only differ in a single literal indicating whether the formula operates in "even mode" or "odd mode."). Since any #SAT problem can be substituted by $2^n - \#\text{DNF}$, as discussed in the introduction, this yields the following characterization (with slight abuse of notation):

$$\#\text{SAT} = \#2\text{SAT} - \#2\text{SAT} = (2^n - \#2\text{DNF}) - (2^n - \#2\text{DNF}) = \#2\text{DNF} - \#2\text{DNF}.$$

This is interesting since the expressive powers of #2SAT and #2DNF are incomparable, unless we obtain[2] RP = NP [27, Theorem 2].

1.2 Simplified Proofs and Consequences Instead of advanced algebraic techniques, our proofs only rely on parity constraints and symmetry. We obtain the following known results as simple consequences of our reduction, as detailed in Section 4.

1. #P-*hardness* of #2SAT. Previously, only complicated reductions were known [28, 29] which went through many problems as described above. Many of these steps were Turing reductions and *only guarantee* polynomially many calls. It turns out that a surprisingly simple #2SAT or #2DNF call is sufficient, that only uses arithmetic post-processing (AC^0).

2. *Efficient algorithm* for #SAT *parameterized by incidence treewidth* k, proving a runtime decrease from $4^k \operatorname{poly}(|\varphi|)$ to $2^k \operatorname{poly}(|\varphi|)$. In contrast to primal treewidth, for which such an algorithm has been known for more than a decade [25], obtaining a 2^k-algorithm for *incidence treewidth had been open for a long time.* In 2020 it was solved with advanced algebraic techniques, e.g., Möbius and Zeta transforms [26].

[2]The result follows from inapproximability of counting independent sets [27, Theorem 2], since independent sets can be trivially counted via #2SAT using one clause per edge, but #2DNF can be efficiently approximated [2].

3. *Sparsification* from $\#d$SAT to sparse $\#$2SAT. Previously, a sparsification lemma from $\#d$SAT to sparse $\#d$SAT has been developed for $d \geq 2$ [10]. Later, this has been extended to establish a sparse lower bound directly for $\#$2SAT, but this relies on Turing reductions and the block interpolation technique [9]. Our approach provides a substantially simpler proof, relies on basic combinatorics, and is based on a many-one reduction plus AC^0 post-processing.

1.3 Related Work Recent work [3] contains more involved proofs and general variants for the presented reduction, but at the cost of losing simplicity. The complexity of $\#$2SAT is an active field of research, dating back to the first proof of its #P-completeness by Valiant [28, 29]. This problem was studied in parameterized settings by Luna et al. [18], and Ita, Bello, and Rodríguez [15]. The problem was also studied in the context of binary patterns [19] and plays an important role in the simulation of quantum systems [17]. There is work on counting complexity for generalized #SAT [13] as well as for using subtraction with counting perfect matchings [8]. Implementing faster algorithms for this fragment of model counting is also an active line of research, e.g., with new bottom-up algorithms [16], or algorithms specialized for formulas with certain structures [20, 21]. There are also proof formats of different expressiveness [4].

1.4 Structure of this Article In the next section, we introduce the necessary notation and provide the simplest version of our reduction, which maps a $\#$SAT instance to a single *weighted* $\#$2SAT simulating the inclusion-exclusion principle. We then illustrate in Section 3 how we can reduce from $\#$SAT the result of two (unweighted) $\#$2SAT instances, which implies the reduction of our main contribution. We discuss various complexity-theoretic and algorithmic implications of our reduction in Section 4 and conclude in Section 5.

2 Warmup: From #SAT to Weighted #2SAT Let us make notions more precise. Unless stated otherwise, we consider *propositional formulas* only in conjunctive normal form (CNF), i.e., as conjunction of *clauses* (disjunctions of literals). For readability, we denote 2-CNF clauses as *implications of the form* $x \to y$, which is equivalent to $\neg x \vee y$. An assignment $\beta \subseteq 2^{\mathrm{vars}(\varphi)}$ is a *set of variables* that are *assigned to true*. A weighted CNF (a WCNF) is a CNF φ together with a mapping $w\colon \mathrm{lits}(\varphi) \to \mathbb{Z}$ that assigns a number to every literal over $\mathrm{vars}(\varphi)$. The *weighted model counting problem* $\#\mathrm{SAT}[\mathbb{Z}]$ asks, given a WCNF (φ, w), to compute the sum of products:

$$\#(\varphi, w) = \sum_{\beta \models \varphi} \prod_{x \in \beta} w(x) \cdot \prod_{y \in (\mathrm{vars}(\varphi) \setminus \beta)} w(\neg y).$$

Here, the sum ranges over the *models (satisfying assignments)* of φ. The *model counting problem* ($\#$SAT) is the special case in which w assigns "1" to very literal, so the above formula simplifies to:

$$\#(\varphi, w) = \#(\varphi) = \sum_{\beta \models \varphi} 1 = |\{\, \beta \mid \beta \models \varphi \,\}|.$$

The basic principle of our reduction from $\#$SAT to $\#$2SAT (evaluate $\#(\varphi)$ for a CNF where every clause contains at most two literals) is visualized via a reduction from $\#$SAT to $\#\mathrm{2SAT}[\mathbb{Z}]$ (compute $\#(\varphi, w)$ for a WCNF in which every clause contains at most two literals). Our construction utilizes the *inclusion-exclusion* principle via the weight function, requiring only weights "1" and "-1" (crucial).

2.1 Simulating Inclusion-Exclusion via Negative Weights Central to our reduction is the idea of reasoning in the inverse, namely, we count *in how many ways an assignment* $\beta \not\models \varphi$ *can falsify clauses.* Whenever this number is even, we will count it positively, while we will subtract it if an odd number of clauses is falsified. Since we count not only the clauses that are indeed falsified, but all the combinations as well, the inclusion-exclusion principle will cancel out these counts and leave us with the number of models of the input formula.

To be a bit more precise, let $\varphi = \bigwedge_{i=1}^{m} c_i$ be the given (unweighted) CNF for which we wish to compute $\#(\varphi)$. Our first goal is to compute a 2-WCNF (ψ, w) such that

$$\#(\varphi) = \#(\psi, w).$$

For *every clause* c *of* φ we introduce a fresh variable $\dot{c}$ with the semantics "we wish to declare c being falsified." The variable $\dot{c}$ will control whether we count clause c for an assignment β that falsifies φ, but $\dot{c}$ is not implied by

the falsification of c. We easily encode this in *formula ψ, which is defined by the following binary clauses*:

$$\boxed{\bigwedge_{\ell \in c} (\dot{c} \to \neg \ell)} \qquad \text{for every clause } c \text{ of } \varphi \tag{2.1}$$

So, if $\dot{c}$ is set to true, all literals in c must be false and, hence, c is falsified. The "trick" is to count the parity of the number of ways we can set $\dot{c}$ variables to true: We introduce *weight function* w by $w(\dot{c}) = -1$ *and* $w(\neg \dot{c}) = 1$ for clauses $c \in \varphi$, and $w(x) = w(\neg x) = 1$ for variables $x \in \text{vars}(\varphi)$. We obtain:

PROPOSITION 2.1. *Let $\varphi = \bigwedge_{i=1}^{m} c_i$ be a* CNF *and $\psi = \bigwedge_{i=1}^{m} \bigwedge_{\ell \in c_i} (\dot{c_i} \to \neg \ell)$ for fresh variables $\dot{c_1}, \ldots, \dot{c_m}$ be a* WCNF *with $w(\dot{c_i}) = -1$ for all $i \in \{1, \ldots, m\}$ and $w(\ell) = 1$ for all other literals. Then $\#(\varphi) = \#(\psi, w)$.*

Proof. Fix any assignment β of $\text{vars}(\varphi)$ and distinguish two cases. In the first case $\beta \models \varphi$ and, thus, β contributes 1 to $\#(\varphi)$. If we extend β to an assignment of ψ, we are forced to set all $\dot{c_i}$ to false (as β does not falsify any clause). Hence, β also *contributes exactly* 1 to $\#(\psi, w)$.

Otherwise, $\beta \not\models \varphi$ and, thus, β does not contribute to $\#(\varphi)$. We can extend β by setting all $\dot{c_i}$ to false (adding $+1$ to $\#(\psi, w)$); but for any clause c_j falsified by β, we can extend β by setting $\dot{c_j}$ to true and all $\dot{c_i}$ with $i \neq j$ to false (subtracting 1 from $\#(\psi, w)$). For each *pair* of falsified clauses, β contributes 1 to $\#(\psi, w)$ (-1's cancel out), for every *triple* we subtract 1 again, and so on. Precisely, if β falsifies $k > 0$ clauses in φ, it *contributes a total of* $0 = 1 + \underbrace{\Sigma_{i=1}^{k} (-1)^i \cdot \binom{k}{i}}_{-1}$ to the count $\#(\psi, w)$. ∎

EXAMPLE 2.2. *Consider formula $\varphi = c_1 \wedge c_2 \wedge c_3$ with $c_1 = (\neg x \vee z)$, $c_2 = (\neg x \vee y \vee z)$, and $c_3 = (y \vee z)$. Below left, we show the truth table of φ revealing $\#(\varphi) = 5$; in the center we show formula ψ of Proposition 2.1. Satisfying assignments like $\beta_2 = \{\neg x, y, \neg z\}$ falsify no clause in φ and only extend to model $\tilde{\beta}_2 = \{\neg x, y, \neg z, \neg \dot{c_1}, \neg \dot{c_2}, \neg \dot{c_3}\}$ of ψ, contributing 1 to weighted count $\#(\psi, w)$. Unsatisfying assignments like $\beta_1 = \{\neg x, \neg y, \neg z\}$ extend to multiple models $\tilde{\beta}_1$ of ψ, totaling to 0 (below right):*

	x	y	z	$\#(\varphi)$
β_1	0	0	0	0
β_2	0	1	0	1
β_3	1	0	0	0
β_4	1	1	0	0
β_5	0	0	1	1
β_6	0	1	1	1
β_7	1	0	1	1
β_8	1	1	1	1
			Total	5

$\dot{c_1} \to x$
$\dot{c_1} \to \neg z$
$\dot{c_2} \to x$
$\dot{c_2} \to \neg y$
$\dot{c_2} \to \neg z$
$\dot{c_3} \to \neg y$
$\dot{c_3} \to \neg z$

Satisfying assignments $\tilde{\beta}_1$ of ψ	$\prod_{\ell \in \tilde{\beta}_1} w(\ell)$
$\neg x, \neg y, \neg z, \neg \dot{c_1}, \neg \dot{c_2}, \neg \dot{c_3}$	1
$\neg x, \neg y, \neg z, \dot{c_1}, \neg \dot{c_2}, \neg \dot{c_3}$	-1
$\neg x, \neg y, \neg z, \neg \dot{c_1}, \dot{c_2}, \neg \dot{c_3}$	-1
$\neg x, \neg y, \neg z, \neg \dot{c_1}, \neg \dot{c_2}, \dot{c_3}$	-1
$\neg x, \neg y, \neg z, \dot{c_1}, \dot{c_2}, \neg \dot{c_3}$	1
$\neg x, \neg y, \neg z, \neg \dot{c_1}, \dot{c_2}, \dot{c_3}$	1
$\neg x, \neg y, \neg z, \dot{c_1}, \neg \dot{c_2}, \dot{c_3}$	1
$\neg x, \neg y, \neg z, \dot{c_1}, \dot{c_2}, \dot{c_3}$	-1
Total	0

From this, we immediately obtain that #2SAT[$\mathbb{Z}$] is #P-complete, even under many-one reductions.

COROLLARY 2.3. *Under parsimonious many-one reductions, #2SAT[$\mathbb{Z}$] is #P-complete.*

3 Main Contribution: From #SAT to #2SAT The inclusion-exclusion reduction above provides us with a blueprint for our reduction. Now, the goal is to reduce #SAT directly to #2SAT. How do we *get rid of these negative weights*? Let again $\varphi = \bigwedge_{i=1}^{m} c_i$ be any given CNF, and let (ψ, w) be the 2-WCNF introduced in and below *Equation (2.1)*. By the design of w, a model of ψ has a weight of $+1$ if the number of $\dot{c_i}$ *set to true is even*, and a weight of -1 otherwise. This leads to the following observation on *translating* #SAT *to* #2SAT:

$$\#(\varphi) = \#(\psi, w) = \#\big(\psi \wedge \neg \bigoplus_{i=1}^{m} \dot{c_i}\big) - \#\big(\psi \wedge \bigoplus_{i=1}^{m} \dot{c_i}\big). \tag{3.1}$$

Note that the two model counts on the right are *unweighted*. That is, to conclude the reduction we only need binary formulas to express $\bigoplus_{i=1}^{m} \dot{c_i}$. For this, we define a *sequential parity counter* next.

3.1 How to Encode Sequential Parity Counters? For the sake of generality, let $\ell_1, \dots, \ell_p$ be any set of ordered literals. We describe a *parity counter formula* $\bigoplus_{i=1}^{p} \ell_i$ by introducing auxiliary variables $e_0, \dots, e_p$ such that e_i indicates that an *even number* of literals up to i is satisfied (including ℓ_i). To implement the semantics of the parity counter, we use further auxiliary variables o_{ℓ_i}, $o_{\bar{\ell}_i}$, e_{ℓ_i}, $e_{\bar{\ell}_i}$ to cover all four cases of having satisfied an *odd number of literals up to* i (due to ℓ_i or $\neg\ell_i$) or an *even number up to* i (due to ℓ_i or $\neg\ell_i$), respectively. The formula $\bigoplus_{i=1}^{p} \ell_i$ contains the clause (e_0) (initially zero literals are satisfied, i.e., an even number) and for every $i \in \{1, \dots, p\}$ we introduce the following set of four (pairwise excluding) sets of clauses:

Case	*Imply the Literal*	*Update the Parity Counter*	
Case 1: Odd due to Literal ℓ_i	$o_{\ell_i} \rightarrow \ell_i$	$o_{\ell_i} \rightarrow e_{i-1}$	$o_{\ell_i} \rightarrow \neg e_i$
Case 2: Odd due to Literal $\neg\ell_i$	$o_{\bar{\ell}_i} \rightarrow \neg\ell_i$	$o_{\bar{\ell}_i} \rightarrow \neg e_{i-1}$	$o_{\bar{\ell}_i} \rightarrow \neg e_i$
Case 3: Even due to Literal ℓ_i	$e_{\ell_i} \rightarrow \ell_i$	$e_{\ell_i} \rightarrow \neg e_{i-1}$	$e_{\ell_i} \rightarrow e_i$
Case 4: Even due to Literal $\neg\ell_i$	$e_{\bar{\ell}_i} \rightarrow \neg\ell_i$	$e_{\bar{\ell}_i} \rightarrow e_{i-1}$	$e_{\bar{\ell}_i} \rightarrow e_i$

In each of the four cases, the first clause states that if we become odd or even by literal ℓ_i (or $\neg\ell_i$), the literal ℓ_i must indeed be satisfied (or falsified, respectively). The last two columns describe how the corresponding case modifies the parity. For instance, in Case 1 we have an odd number of satisfied literals up to i (including ℓ_i) since ℓ_i is satisfied. Hence, up to $i-1$ we must have seen an even number of satisfied literals (second clause in this row) and now we are odd, i.e., not even (last clause).

DEFINITION 3.1 (Sequential Parity Counter Encoding). *For ordered literals $\ell_1, \dots, \ell_p$, we define*

$$\begin{aligned}\neg \bigoplus_{i=1}^{p} \ell_i \quad = \quad & e_0 \wedge e_p \wedge \bigwedge_{i=1}^{p} \Big[(o_{\ell_i} \rightarrow \ell_i) \wedge (o_{\ell_i} \rightarrow e_{i-1}) \wedge (o_{\ell_i} \rightarrow \neg e_i) \\ & \wedge (o_{\bar{\ell}_i} \rightarrow \neg\ell_i) \wedge (o_{\bar{\ell}_i} \rightarrow \neg e_{i-1}) \wedge (o_{\bar{\ell}_i} \rightarrow \neg e_i) \\ & \wedge (e_{\ell_i} \rightarrow \ell_i) \wedge (e_{\ell_i} \rightarrow \neg e_{i-1}) \wedge (e_{\ell_i} \rightarrow e_i) \\ & \wedge (e_{\bar{\ell}_i} \rightarrow \neg\ell_i) \wedge (e_{\bar{\ell}_i} \rightarrow e_{i-1}) \wedge (e_{\bar{\ell}_i} \rightarrow e_i) \\ & \wedge (\mathbf{o_{\ell_i}} \vee \mathbf{o_{\bar{\ell}_i}} \vee \mathbf{e_{\ell_i}} \vee \mathbf{e_{\bar{\ell}_i}}) \Big].\end{aligned}$$

Furthermore, $\bigoplus_{i=1}^{p} \ell_i$ is the same formula as $\neg \bigoplus_{i=1}^{p} \ell_i$, but with the singleton "e_p" clause negated.

LEMMA 3.2 (Correctness of Parity Counter Encoding). *Let $\ell_1, \dots, \ell_p$ be ordered literals. Then, we have $\neg \bigoplus_{i=1}^{p} \ell_i \equiv \neg(\ell_1 \oplus \ell_2 \oplus \cdots \oplus \ell_p)$ and $\bigoplus_{i=1}^{p} \ell_i \equiv (\ell_1 \oplus \ell_2 \oplus \cdots \oplus \ell_p)$, where $(a \oplus b)$ is an* exclusive-or, *as defined in the usual meaning by $(a \vee b) \wedge (\neg a \vee \neg b)$.*

Proof. The proof is by induction over i (showing that the xor of the first i literals is computed correctly) with the induction hypothesis given by the trivial case $i = 0$ (i.e., the parity counter has not yet processed any literal), which is correctly realized as e_0 is a clause. For the induction step, consider any $i > 0$ and distinguish the two cases that ℓ_i is true or false. If ℓ_i is true, only Cases 1 and 3 can be relevant (i.e., only o_{ℓ_i} or e_{ℓ_i} can be set to true), as the other cases imply $\neg\ell_i$ and thus yield a contradiction. By the induction hypothesis, we can assume that e_{i-1} is set correctly. Assume without loss of generality that e_{i-1} is true (the other case is symmetric) and observe that, therefore, only Case 1 is applicable (Case 3 implies $\neg e_{i-1}$). Hence, the counter correctly implies $\neg e_i$. Note that by the *red clause (clause of size four)*, at least one case must be selected.

The other two cases (becoming odd or even by $\neg\ell_i$) work identically and we can, thus, conclude that the counter correctly sets e_i to true if and only if an even number of the literals $\ell_1, \dots, \ell_i$ is true. Both statements of the theorem follow immediately. ∎

This lemma alone does not help much due to the *red clause*, which is *clearly not binary*. However, it turns out — almost magically — that if we utilize the lemma in Equation (3.1), we can *drop the red clauses* and still obtain the correct result! Although the parity counter will, of course, make errors without the red clause, it will make exactly the same number of errors in both cases and, therefore, these errors cancel out in Equation (3.1).

3.2 Reduction to #2SAT More precisely, our reduction takes φ and works as follows. First, we construct ψ as in *Equation (2.1)*. Then, we define

$$\boxed{\psi_1 = \psi \wedge \xi \qquad \text{and} \qquad \psi_2 = \psi \wedge \xi',} \tag{3.2}$$

where ξ is the party counter $\neg \bigoplus_{i=1}^{|\varphi|} \dot{c}_i$ *from Definition 3.1* but *without the red clause*, and ξ' is $\bigoplus_{i=1}^{|\varphi|} \dot{c}_i$ *without the red clause*. It remains to prove:

$$\boxed{\#(\varphi) = \#(\psi_1) - \#(\psi_2).} \tag{3.3}$$

Observe that both ψ_1 and ψ_2 are almost identical and differ only in a single clause (a single literal). We show that because of this symmetry, the unwanted models of ψ_1 and ψ_2 systematically cancel each other out. We refer to these unwanted models by *rogue*, which are defined as follows.

DEFINITION 3.3 (Rogue Models). *Let φ be a formula and let ψ_1, ψ_2 be constructed as above. A model β of ψ_1 or ψ_2 is* rogue *at $1 \leq i \leq |\varphi|$ if $\beta \cap \{o_{c_i}, o_{\overline{c}_i}, e_{c_i}, e_{\overline{c}_i}\} = \emptyset$, i.e., if it violates* the red clause.

3.3 Correctness by Symmetry Indeed, we can establish a bijection between these rogue models, if defined carefully. This is ensured by the following construction, which uniquely defines for every rogue model of ψ_1 the corresponding unique symmetric rogue model of ψ_2. Intuitively, if a model is rogue at some i, we do not decide for a case, so we can freely switch the parity merit at i (swapping even by odd and vice versa).

DEFINITION 3.4 (Symmetric Rogue Model). *Let φ be a formula and β be a model of ψ_1 (ψ_2) that is rogue at some largest $1 \leq i \leq |\varphi|$, so β is not rogue at any $j > i$. Then the symmetric rogue model β' of ψ_2 (ψ_1) is obtained from β by (1)* swapping parity*: replacing for every $i' \geq i$, $e_{i'} \in \beta$ by $e_{i'} \notin \beta'$ and vice versa: $e_{i'} \notin \beta$ by $e_{i'} \in \beta'$; (2)* swapping parity of case*: replacing for every $j > i$:*

- *if o_{c_j} is in β, then replace o_{c_j} by e_{c_j} in β';*
- *if e_{c_j} is in β, then replace e_{c_j} by o_{c_j} in β';*
- *if $o_{\overline{c}_j}$ is in β, then replace $o_{\overline{c}_j}$ by $e_{\overline{c}_j}$ in β';*
- *if $e_{\overline{c}_j}$ is in β, then replace $e_{\overline{c}_j}$ by $o_{\overline{c}_j}$ in β'.*

Before we show correctness, we will briefly clarify these concepts using our running example.

EXAMPLE 3.5. *Recall the formula $\varphi = c_1 \wedge c_2 \wedge c_3$ with $c_1 = (\neg x \vee z)$, $c_2 = (\neg x \vee y \vee z)$, and $c_3 = (y \vee z)$ from the previous example. As we have seen, formula ψ according to Equation (2.1) is:*

$$\psi = (\dot{c}_1 \to x) \wedge (\dot{c}_1 \to \neg z) \wedge (\dot{c}_2 \to x) \wedge (\dot{c}_2 \to \neg y) \wedge (\dot{c}_2 \to \neg z) \wedge (\dot{c}_3 \to \neg y) \wedge (\dot{c}_3 \to \neg z).$$

The encoding of $\bigoplus_{i=1}^{3} \dot{c}_i$ due to Definition 3.1 is given by $e_0 \wedge \neg e_p$ and the following set of clauses:

$$\underbrace{\begin{array}{cccc} o_{\dot{c}_1} \to \dot{c}_1 & o_{\dot{c}_1} \to e_0 & o_{\dot{c}_1} \to \neg e_1 \\ o_{\overline{\dot{c}_1}} \to \neg \dot{c}_1 & o_{\overline{\dot{c}_1}} \to \neg e_0 & o_{\overline{\dot{c}_1}} \to \neg e_1 \\ e_{\dot{c}_1} \to \dot{c}_1 & e_{\dot{c}_1} \to \neg e_0 & e_{\dot{c}_1} \to e_1 \\ e_{\overline{\dot{c}_1}} \to \neg \dot{c}_1 & e_{\overline{\dot{c}_1}} \to e_0 & e_{\overline{\dot{c}_1}} \to e_1 \\ \multicolumn{3}{c}{(\mathbf{o}_{\dot{c}_1} \vee \mathbf{o}_{\overline{\dot{c}_1}} \vee \mathbf{e}_{\dot{c}_1} \vee \mathbf{e}_{\overline{\dot{c}_1}})} \end{array}}_{i=1} \quad \underbrace{\begin{array}{ccc} o_{\dot{c}_2} \to \dot{c}_2 & o_{\dot{c}_2} \to e_1 & o_{\dot{c}_2} \to \neg e_2 \\ o_{\overline{\dot{c}_2}} \to \neg \dot{c}_2 & o_{\overline{\dot{c}_2}} \to \neg e_1 & o_{\overline{\dot{c}_2}} \to \neg e_2 \\ e_{\dot{c}_2} \to \dot{c}_2 & e_{\dot{c}_2} \to \neg e_1 & e_{\dot{c}_2} \to e_2 \\ e_{\overline{\dot{c}_2}} \to \neg \dot{c}_2 & e_{\overline{\dot{c}_2}} \to e_1 & e_{\overline{\dot{c}_2}} \to e_2 \\ \multicolumn{3}{c}{(\mathbf{o}_{\dot{c}_2} \vee \mathbf{o}_{\overline{\dot{c}_2}} \vee \mathbf{e}_{\dot{c}_2} \vee \mathbf{e}_{\overline{\dot{c}_2}})} \end{array}}_{i=2} \quad \underbrace{\begin{array}{ccc} o_{\dot{c}_3} \to \dot{c}_3 & o_{\dot{c}_3} \to e_2 & o_{\dot{c}_3} \to \neg e_3 \\ o_{\overline{\dot{c}_3}} \to \neg \dot{c}_3 & o_{\overline{\dot{c}_3}} \to \neg e_2 & o_{\overline{\dot{c}_3}} \to \neg e_3 \\ e_{\dot{c}_3} \to \dot{c}_3 & e_{\dot{c}_3} \to \neg e_2 & e_{\dot{c}_3} \to e_3 \\ e_{\overline{\dot{c}_3}} \to \neg \dot{c}_3 & e_{\overline{\dot{c}_3}} \to e_2 & e_{\overline{\dot{c}_3}} \to e_3 \\ \multicolumn{3}{c}{(\mathbf{o}_{\dot{c}_3} \vee \mathbf{o}_{\overline{\dot{c}_3}} \vee \mathbf{e}_{\dot{c}_3} \vee \mathbf{e}_{\overline{\dot{c}_3}})} \end{array}}_{i=3}$$

We obtain the following counts:

$$\#\big(\psi \wedge \neg \bigoplus_{i=1}^{3} \dot{c}_i\big) = 11 \qquad \#\big(\psi \wedge \bigoplus_{i=1}^{3} \dot{c}_i\big) = 6 \qquad \#\big(\psi \wedge \neg \bigoplus_{i=1}^{3} \dot{c}_i\big) - \#\big(\psi \wedge \bigoplus_{i=1}^{3} \dot{c}_i\big) = \underline{\underline{11 - 6 = 5}} = \#(\varphi).$$

Let now ξ and ξ' be the encodings of $\neg \bigoplus_{i=1}^{3} \dot{c}_i$ and $\bigoplus_{i=1}^{3} \dot{c}_i$ without the red clauses*, respectively. Then, by systematic analysis or utilizing code writing talent, we obtain:*

$$\#\big(\psi \wedge \xi\big) = 232 \qquad \#\big(\psi \wedge \xi'\big) = 227 \qquad \#\big(\psi \wedge \xi\big) - \#\big(\psi \wedge \xi'\big) = \underline{\underline{232 - 227 = 5}} = \#(\varphi).$$

It turns out that these definitions are sufficient to show that our reduction is correct, i.e., that Equation (3.3) holds. This is established via the following two lemmas, which we use to prove correctness.

LEMMA 3.6 (Non-Rogue Models). *Let φ be a formula and let ψ_1, ψ_2 be defined as before. We have:*

1. *For every model β of ψ_1 or ψ_2 that is not rogue, $\beta \cap \text{vars}(\varphi)$ is an interpretation of φ that invalidates at least $k \coloneqq |\{\, c \in \varphi \mid \dot{c} \in \beta \,\}|$ clauses.*
2. *The value k is odd iff β is a model of ψ_2.*

Proof. By the construction of ψ from φ in Equation (2.1), we know that β invalidates (at least) k clauses. By Lemma 3.2 and since β is not rogue, k is odd (even) iff β is a model of ψ_2 (ψ_1), respectively. ∎

LEMMA 3.7 (Well-Defined: Rogue Bijection). *Let φ be a formula and let ψ_1, ψ_2 be defined as before. Then, the symmetric rogue model β' of any rogue model β of ψ_1 is a model of ψ_2 (and vice versa). Further, the symmetric rogue model of the symmetric rogue model β' of ψ_1 (ψ_2) is β.*

Proof. Let β be rogue at $1 \leq i \leq |\varphi|$ such that there is no $i < j \leq |\varphi|$ at which β is rogue. By definition, we have $\beta \cap \{o_{c_i}, o_{\bar{c}_i}, e_{c_i}, e_{\bar{c}_i}\} = \emptyset$. Note that therefore, by construction of ψ_1, we can switch the parity at i, but since there is no $j > i$ where β is rogue at j, all parities at $j > i$ are determined accordingly. So we can construct β' from β, where for every $i' \geq i$ we (i) replace $e_{i'} \in \beta$ with $e_{i'} \notin \beta'$ and (ii) replace $e_{i'} \notin \beta$ with $e_{i'} \in \beta'$. This just swaps the merit of parity, but we also need to swap the reason, so for every $j > i$, we swap $o_{c_j} \in \beta$ by $e_{c_j} \in \beta'$, $o_{\bar{c}_j} \in \beta$ by $e_{\bar{c}_j} \in \beta'$, $e_{c_j} \in \beta$ by $o_{c_j} \in \beta'$, and $e_{\bar{c}_j} \in \beta$ by $o_{\bar{c}_j} \in \beta'$. Indeed, this is the result of ensuring consistency in β' due to (i) and (ii) above. To sum up, β' is precisely defined as

$$\begin{aligned}\beta' = &\left[\beta \cap (\text{vars}(\psi_1) \setminus \{e_{i'}, o_{c_{i'}}, o_{\bar{c}_{i'}}, e_{c_{i'}}, e_{\bar{c}_{i'}} \mid i' \geq i\})\right] \\ &\cup \{e_{i'} \mid e_{i'} \notin \beta, i \leq i' \leq |\varphi|\} \\ &\cup \{o_{c_j} \mid e_{c_j} \in \beta, j > i\} \cup \{o_{\bar{c}_j} \mid e_{\bar{c}_j} \in \beta, j > i\} \\ &\cup \{e_{c_j} \mid o_{c_j} \in \beta, j > i\} \cup \{e_{\bar{c}_j} \mid o_{\bar{c}_j} \in \beta, j > i\}.\end{aligned}$$

It is easy to see that, therefore, the final parity at $|\varphi|$ switches from even to odd when comparing β with β'. Consequently, by the symmetric construction of ψ_1 and ψ_2 the result follows. Indeed, by the construction in Definition 3.4, the symmetric rogue model of the symmetric rogue model β' is β. ∎

Both lemmas are sufficient to establish correctness, which we show below.

THEOREM 3.8 (Correctness). *Equation (3.3) is correct for any formula φ, i.e., $\#(\varphi) = \#(\psi_1) - \#(\psi_2)$.*

Proof. The construction simulates the principle of inclusion-exclusion. We count the models of φ as:

$$\#(\varphi) = 2^n - \sum_{\substack{\beta \subseteq \text{vars}(\varphi), \exists c \in \varphi, \\ \beta \not\models \{c\}}} 1 + \sum_{\substack{\beta \subseteq \text{vars}(\varphi), \exists c, c' \in \varphi, \\ c \neq c', \beta \not\models \{c\} \text{ and } \beta \not\models \{c'\}}} 1 \quad \dots \quad \sum_{\substack{\beta \subseteq \text{vars}(\varphi), \\ \nexists c \in \varphi, \beta \models \{c\}}} (-1)^{|\varphi|} = \sum_{\substack{\beta \subseteq \text{vars}(\varphi), 0 \leq k \leq |\varphi|, \\ \beta \text{ does not satisfy} \geq k \text{ clauses in } \varphi}} (-1)^k.$$

Therefore, we split this term into two parts, where we take interpretations not satisfying at least k clauses with k being even (ψ_1) and then subtract those interpretations with k being odd (ψ_2). We refer to those interpretations where k is even by

$$E = \{\beta \subseteq \text{vars}(\varphi) \mid 0 \leq n \leq |\varphi|, \beta \text{ does not satisfy} \geq k \text{ clauses in } \varphi, k \equiv 0 (\text{mod} 2)\}.$$

The interpretations with k being odd are referred to by

$$O = \{\beta \subseteq \text{vars}(\varphi) \mid 0 \leq k \leq |\varphi|, \beta \text{ does not satisfy} \geq k \text{ clauses in } \varphi, k \equiv 1 (\text{mod} 2)\}.$$

By Lemma 3.6, every model in E can be uniquely extended to a non-rogue model of ψ_1. By Lemma 3.2, for every non-rogue model of ψ_1, its restriction to $\text{vars}(\varphi)$ is in E. In addition, models of ψ_1 are either rogue (say, in R) or in E, i.e., $|E| + |R| = \#(\psi_1)$. Analogously, for ψ_2 we can show that $|O| + |R'| = \#(\psi_2)$, for some set R'. By Lemma 3.7, there is a bijection between the rogue models R of ψ_1 and the rogue models R' of ψ_2. Consequently, $\#(\varphi) = |E| - |O| = \big(|E| + |R|\big) - \big(|O| + |R'|\big) = \#(\psi_1) - \#(\psi_2)$. ∎

4 Complexity-Theoretic and Algorithmic Implications From the simple construction presented in the previous section, we immediately obtain simpler, more fine-grained precise hardness proofs for counting on 2-CNF or 2-DNF formulas.

4.1 Simplified Hardness In contrast to Corollary 2.3, our reduction must not be parsimonious (unless NL = NP). However, a *single call is still sufficient* (even with very limited post-processing).

COROLLARY 4.1 (#2SAT Hardness). *#2SAT is #P-hard. This even holds for a single #2SAT instance with* AC^0 *post-processing power.*

Proof. The first part follows from the correctness of the #SAT reduction to #2SAT as shown in Theorem 3.8, since a single #3SAT instance can parsimoniously simulate a Turing machine [28, Lemma 3.2].

The second part is a simple extension. Without loss of generality, ψ_1 and ψ_2 *do not share variables*, which works by renaming variables from v to v' in ψ_2. We merge both formulas and use fresh variable s to create clauses $s \to v$ and $\neg s \to v'$ for every $v \in \mathrm{vars}(\psi_1) \setminus \{e_0, e_{|\varphi|}\}$ and $v' \in \mathrm{vars}(\psi_2) \setminus \{e'_0, e'_{|\varphi|}\}$. This ensures that if $\neg s$ holds, we obtain models of ψ_1 and if s holds, we obtain models of ψ_2.

Further, we use $|\mathrm{vars}(\psi_1)|+1$ fresh variables a_j, and clauses $s \to \neg a_j$ for every $1 \leq j \leq |\mathrm{vars}(\psi_1)|+1$. This results in multiplying every model of ψ_1 by a factor $2^{|\mathrm{vars}(\psi_1)|+1}$, as variables a_j are free if $\neg s$ holds. Hence, the resulting count is of the form $c_1 \cdot 2^{|\mathrm{vars}(\psi_1)|+1} + c_2$, where $c_1 = \#(\psi_1)$ and $c_2 = \#(\psi_2)$. We extract c_1 and c_2 by shifting, followed by subtraction $c_1 - c_2$, both of which work in AC^0. ∎

Since #2DNF is the inverse problem of #2SAT, in Equation (3.3), we can also replace $\#(\psi_1) - \#(\psi_2)$ by $[2^{\mathrm{vars}(\psi_1)} - \#(\neg\psi_1)] - [2^{\mathrm{vars}(\psi_2)} - \#(\neg\psi_2)]$, which simplifies to $\#(\neg\psi_2) - \#(\neg\psi_1)$.

COROLLARY 4.2 (#2DNF Hardness). *#2DNF is #P-hard, even for a single #2DNF instance with* AC^0 *post-processing power.*

4.2 Simplified Treewidth-Based Algorithm Given the simplicity of our reduction, we can easily extend it to obtain efficient upper bounds for treewidth that are tight assuming the *strong exponential time hypothesis (*SETH*)* [14]. The runtime bound of the resulting algorithm is a corollary of our reduction and subsumes a recent paper [26] that uses algebraic techniques based on the Möbius and Zeta transforms, and covering products.

A *tree decomposition* (T, χ) of a graph G is a rooted tree T and a mapping $\chi : V(T) \to 2^{V(G)}$, called *bag*, such that (1) for every $v \in V(G)$ the set $\{\, x \mid v \in \chi(x) \,\}$ is non-empty and connected in T; (2) for every $\{u, v\} \in E(G)$ there is a node $x \in V(T)$ with $\{u, v\} \subseteq \chi(x)$. The *width* of a tree decomposition relates to the maximum bag size, i.e., $\max_{x \in V(T)} |\chi(x)| - 1$. The *treewidth* of G is the minimum width among every decomposition of G; $\mathrm{children}(t)$ is the set of child nodes of a node t in T.

We observe that while the reduction and correctness of Section 3.2 are carried out along a path, they can easily be extended to tree decompositions. For this we need the *incidence graph* I_φ of a CNF formula φ, which is a bipartite graph whose vertices are for clauses and variables of φ with an edge between a variable vertex v and a clause vertex $\dot{c}$, whenever variable x appears in clause c. For φ we let $\mathrm{itw}(\varphi)$ be the *treewidth of* I_φ. We obtain the following parameter-awareness result of our reduction.

LEMMA 4.3 (Treewidth-Aware Reduction). *There is a reduction that reduces any* CNF *formula* φ *to* 2-CNF *(*2-DNF*) formulas* ψ_1, ψ_2 *such that* $\#(\varphi) = \#(\psi_1) - \#(\psi_2)$ *and we additively preserve incidence treewidth, i.e.,* $\mathrm{itw}(\psi_1) = \mathrm{itw}(\psi_2) \leq \mathrm{itw}(\varphi) + 8$.

Proof. Without loss of generality, we may assume a tree decomposition $\mathcal{T} = (T, \chi)$ of the incidence graph I_φ of size linear in $|\varphi|$, where nodes have at most two child nodes (these nodes t are called *join nodes*; their $\chi(t)$ are identical to both child bags of t) in T [6].

Then, each clause $c \in \varphi$ can be *assigned to a unique non-join node* $t = \delta(c)$ in T containing $\dot{c} \in \chi(t)$ ($\chi(t)$ might not contain all variables of c). Indeed, if there are not enough decomposition nodes, one can easily add intermediate copy nodes of a node with identical bags, allowing to assign at most one clause per node. Decomposition nodes (except join nodes) that do not have an assigned clause can be assigned a fresh dummy clause $c' = x' \vee x'$ for some fresh x', which increases bag sizes by at most +2. We assume that join nodes do not get assigned such a clause, which will make the construction easier.

However, for *join nodes* t *we slightly modify* the four cases constructed in the reduction (see Section 3.2 and Definition 3.1). Instead of case decision based on clause variables $\dot{c}$, we decide based on the parity of t and the

parity of the first (left-most) child node t'; from this we infer parity of the other child node t'', which then still requires four cases.

Case	*Imply the Literal*	*Update the Parity Counter*	
Case 1: Odd due to Literal $e_{t'}$	$o_{e_{t'}} \rightarrow e_{t'}$	$o_{e_{t'}} \rightarrow \neg e_{t''}$	$o_{e_{t'}} \rightarrow \neg e_t$
Case 2: Odd due to Literal $\neg e_{t'}$	$o_{\bar{e}_{t'}} \rightarrow \neg e_{t'}$	$o_{\bar{e}_{t'}} \rightarrow e_{t''}$	$o_{\bar{e}_{t'}} \rightarrow \neg e_t$
Case 3: Even due to Literal $e_{t'}$	$e_{e_{t'}} \rightarrow e_{t'}$	$e_{e_{t'}} \rightarrow e_{t''}$	$e_{e_{t'}} \rightarrow e_t$
Case 4: Even due to Literal $\neg e_{t'}$	$e_{\bar{e}_{t'}} \rightarrow \neg e_{t'}$	$e_{\bar{e}_{t'}} \rightarrow \neg e_{t''}$	$e_{\bar{e}_{t'}} \rightarrow e_t$

For showing correctness, it suffices to consider *some canonical root-to-leaf path* (say the left-most one) in the tree decomposition that has a node t where the rogue model M is rogue at t. Note that this is not a static path, but the smallest one with a node t where M is rogue at t. Then, the symmetric rogue model of Definition 3.4 on tree decompositions considers for every rogue model its canonical path; correctness works analogously to the proof of Theorem 3.8.

It remains to show *treewidth preservation*, which we can easily demonstrate by constructing from $\mathcal{T}$ a tree decomposition $\mathcal{T}' = (T', \chi')$ of I_{ψ_1} (I_{ψ_2}). Intuitively, the fresh clauses we construct (say, p many) are only over variables in bags and child bags in T; T' is obtained from T by replacing every node in t by a path $t_1, \ldots, t_p$ comprising of p copies of t. Thus, each freshly constructed clause c'_i can be placed in one of these copies. Formally, for every node t in T and $i \in \{1, \ldots, p\}$ we define

$$\begin{aligned}\chi'(t_i) = \chi(t) &\cup \{c'_i, e_t\} \\ &\cup \{e_{t'} \mid t' \in \text{children}(t)\} \cup \{e_c, e_{\bar{c}}, o_c, o_{\bar{c}} \mid c \in \chi(t) \text{ where } t = \delta(c)\} \\ &\cup \{e_{e_{t'}}, e_{\bar{e}_{t'}}, o_{e_{t'}}, o_{\bar{e}_{t'}} \mid |\text{children}(t)| = 2 \text{ and } t' \text{ is the left-most child of } t\}.\end{aligned}$$

Therefore, $|\chi'(t_i)| \leq |\chi(t)| + 2 + 2 + 4$, concluding the proof. ∎

This is already enough to obtain an algorithm that runs in $2^k \operatorname{poly}(|\varphi|)$ for the incidence treewidth k, as we go via a known algorithm for the primal graph. The *primal graph* P_φ of φ has as vertices the variables $\operatorname{vars}(\varphi)$ with an edge between two variables if they appear in a clause in φ; the *primal treewidth* $\operatorname{tw}(\varphi)$ corresponds to the treewidth of P_φ.

Corollary 4.4 (SAT Incidence Treewidth Runtime 2^k). #SAT *for formulas φ can be solved in time $2^k \cdot \operatorname{poly}(|\varphi|)$, where $k = \operatorname{itw}(\varphi)$ is the treewidth of the incidence graph of φ.*

Proof. By Lemma 4.3, we can reduce a CNF φ to 2-CNFs ψ_1 and ψ_2 with $\operatorname{itw}(\psi_i) \leq \operatorname{itw}(\varphi) + 8$. In the incidence graph of a 2-CNF formula, all vertices corresponding to clauses have maximum degree 2. By the *almost simplicial rule*, contracting such a vertex to one of its neighbors cannot increase the treewidth past 2 [7]. However, contracting all vertices corresponding to a clause to one of their neighbors yields exactly the primal graph, hence, we have $\operatorname{tw}(\psi_i) \leq \operatorname{itw}(\psi_i) + 1 \leq \operatorname{itw}(\varphi) + 9$. Finally, we compute $\#(\psi_i)$ by dynamic programming over a tree decomposition of the primal graph, requiring $O\big(2^{\operatorname{tw}(\psi_i)}|\psi_i|\big)$ arithmetic operations [5, 25]. ∎

4.3 Sparsification and Lower Bound for #2SAT Many #P-hardness reductions from #SAT increase size by a linear factor of $n + m$ where n is the number of variables and m is the number of clauses. There, if we want to preserve lower bounds, we must get a $2^{o(n)}$ bound even when $m = \mathcal{O}(n)$; we call instances with this property *sparse instances*. It turns out that our approach sketched above preserves sparsity. This allows us to define a sparsification lemma for #2SAT and #2DNF, which improves an existing sparsification result [10, Lemma A.1].

By applying the simple symmetric reduction sketched in this paper, we obtain a fine-grained sparsification for counting. This provides a sparsification corollary for counting, which is not based on Turing reductions and also does *not require advanced techniques like block interpolation* [9]. Block interpolation is a special kind of *multi-call Turing reduction* which (1) uses an oracle to evaluate some multivariate polynomial $f(x)$ on $n+1$ points and (2) use Lagrange interpolation to get the coefficients of the function. Thus in order to use this technique we must design some sets of gadgets to reduce each point to a different instance of #2SAT, make $n+1$ oracle calls, then perform complicated arithmetic with the answers. Instead, we obtain a variant that, together with Corollary 4.1, only reduces to sparse counting with AC^0 post-processing.

COROLLARY 4.5 (Sparsification Corollary for #2SAT/#2DNF). *Let $d \geq 2$. For every $k \in \mathbb{N}$ and d-CNF formula γ with n variables, there exists $t \in \mathbb{N}$ such that in time $t \cdot poly(n)$ we obtain formulas γ_1^i and γ_2^i for every $i \in [t]$ and*

1. $\#(\gamma) = \sum_{i \in [t]} [\#(\gamma_1^i) - \#(\gamma_2^i)]$,
2. $t \leq 2^{\frac{n}{k}}$,
3. γ_1^i, γ_2^i *are* 2-CNF *(*2-DNF*) formulas in which each variable occurs at most* 3 *times.*

Proof. By [10, Lem. A.1], there is a formula $\beta = \bigvee_{i \in [t]} \gamma_i$ such that $\#(\gamma) = \sum_{i \in [t]} \#(\gamma_i)$ and γ_i in 3-CNF. Claims (1), (3) are modifications of [10, Lem. A.1] that follow from applying the reduction of Section 3.2 on every γ^i, which is correct by Theorem 3.8. The restriction to degree 3 in the 2-CNF easily works with copies x_i for every variable x and known 2-CNF connector gadgets [24] that are cycles of the form $x \to x_1$, $x_1 \to x_2$, ..., $x_k \to x$. Indeed, this enforces equivalence among all copies and each copy can now host one original clause. The 2-DNF claim follows by the observation above Corollary 4.2. ∎

This result immediately provides the following sparse lower bound based on #ETH [10] without the need for block interpolation [9], additionally improving from degree $\mathcal{O}(1)$ to degree 3.

COROLLARY 4.6 (#ETH LB for #2SAT). *Under* #ETH *we cannot solve* #2SAT *in time $2^{o(n)} \cdot poly(n)$, where n is the number of variables, variables occur ≤ 3 times, and even with $m = \mathcal{O}(n)$ clauses.*

Proof. The result follows from Corollary 4.5 by reducing from #3SAT, which under #ETH cannot be solved in time $2^{o(n)} \cdot \text{poly}(n)$. Indeed, a $2^{o(m)} \cdot \text{poly}(n)$ runtime for #2SAT would contradict #ETH. ∎

It turns out that the construction in Section 3 can be extended to monotone #2SAT as well as to monotone #2DNF. However, this requires extensions of the concept of rogue models, which results in a more involved construction of the symmetric rogue model, but comes at the cost of losing simplicity [3].

5 Conclusion and Outlook In this article, we presented a new, remarkably simple reduction from #SAT to two calls of #2SAT, thus providing a simplified proof that the latter problem is #P-complete under Turing reductions. The main idea of our reduction was to use a pipeline that "almost" works — namely, first reducing to weighted #2SAT and then shifting the weights into subtraction using parity constraints. While this is a straightforward construction, it only "almost" works because the parity constraints apparently require at least some non-binary clauses. In fact, we are not aware of any direct technique to implement these parity constraints without the introduction of these clauses. The main "trick" of our new proof is the observation that, since both formulas are almost identical, these non-binary clauses can simply be *omitted.* While this introduces bugs in the parity constraints (they *will overcount*), these bugs are "symmetric" in both formulas in the sense that both *over count by exactly the same amount* — hence, the errors we introduce by omitting the non-binary clauses cancel out.

The simplicity of the construction also unlocks various corollaries, or simplifies the original proofs of them. First, since our reduction is logspace and linear time computable, it strengthens the formulas for which #2SAT is already hard for #P. Second, it is relatively easy to see that our reduction preserve all kinds of structural parameters of the input. In particular, our reduction preserves the input's *incidence treewidth,* leading to a new and simple proof of the fact that #SAT can be solved in time $2^k \text{poly}(|\varphi|)$ on formulas of incidence treewidth k. Finally, we obtained an improved version of the sparsification lemma from #dSAT to sparse #2SAT.

References

[1] C. ÀLVAREZ AND B. JENNER, *A Very Hard Logspace Counting Class*, Theor. Comput. Sci., 107 (1993), pp. 3–30, https://doi.org/10.1016/0304-3975(93)90252-O.

[2] M. ARENAS, L. A. CROQUEVIELLE, R. JAYARAM, AND C. RIVEROS, *#NFA Admits an FPRAS: Efficient Enumeration, Counting, and Uniform Generation for Logspace Classes*, J. ACM, 68 (2021), pp. 48:1–48:40, https://doi.org/10.1145/3477045.

[3] M. BANNACH, E. D. DEMAINE, T. GOMEZ, AND M. HECHER, *#P is Sandwiched by One and Two #2DNF Calls: Is Subtraction Stronger Than We Thought?*, in 40th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS 2025), IEEE, 2025, pp. 31–43, https://doi.org/10.1109/LICS65433.2025.00010. In Press. Available at https://arxiv.org/abs/2506.06716.

[4] O. Beyersdorff, J. K. Fichte, M. Hecher, T. Hoffmann, and K. Kasche, *The Relative Strength of #SAT Proof Systems*, in 27th International Conference on Theory and Applications of Satisfiability Testing (SAT 2024), S. Chakraborty and J. R. Jiang, eds., vol. 305 of LIPIcs, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024, pp. 5:1–5:19, https://doi.org/10.4230/LIPICS.SAT.2024.5.

[5] H. L. Bodlaender, P. S. Bonsma, and D. Lokshtanov, *The Fine Details of Fast Dynamic Programming over Tree Decompositions*, in 8th International Symposium on Parameterized and Exact Computation (IPEC 2013), 2013, pp. 41–53, https://doi.org/10.1007/978-3-319-03898-8_5.

[6] H. L. Bodlaender and T. Kloks, *Efficient and Constructive Algorithms for the Pathwidth and Treewidth of Graphs*, Journal of Algorithms, 21 (1996), pp. 358–402.

[7] H. L. Bodlaender, A. M. C. A. Koster, F. van den Eijkhof, and L. C. van der Gaag, *Pre-processing for Triangulation of Probabilistic Networks*, in 17th Conference in Uncertainty in Artificial Intelligence (UAI 2001), 2001, pp. 32–39.

[8] R. Curticapean, *Parity Separation: A Scientifically Proven Method for Permanent Weight Loss*, in 43rd International Colloquium on Automata, Languages, and Programming (ICALP 2016), I. Chatzigiannakis, M. Mitzenmacher, Y. Rabani, and D. Sangiorgi, eds., vol. 55 of LIPIcs, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016, pp. 47:1–47:14, https://doi.org/10.4230/LIPICS.ICALP.2016.47.

[9] R. Curticapean, *Block Interpolation: A Framework for Tight Exponential-Time Counting Complexity*, Inf. Comput., 261 (2018), pp. 265–280, https://doi.org/10.1016/J.IC.2018.02.008.

[10] H. Dell, T. Husfeldt, D. Marx, N. Taslaman, and M. Wahlen, *Exponential Time Complexity of the Permanent and the Tutte Polynomial*, ACM Trans. Algorithms, 10 (2014), pp. 21:1–21:32, https://doi.org/10.1145/2635812.

[11] M. E. Dyer, L. A. Goldberg, C. S. Greenhill, and M. Jerrum, *The Relative Complexity of Approximate Counting Problems*, Algorithmica, 38 (2004), pp. 471–500, https://doi.org/10.1007/S00453-003-1073-Y.

[12] J. K. Fichte, M. Hecher, and F. Hamiti, *The Model Counting Competition 2020*, ACM J. Exp. Algorithmics, 26 (2021), pp. 13:1–13:26, https://doi.org/10.1145/3459080.

[13] M. H. Group, J. Brunner, E. D. Demaine, J. Diomidova, T. Gomez, M. Hecher, F. Stock, and Z. Zhou, *Easier Ways to Prove Counting Hard: A Dichotomy for Generalized #SAT, Applied to Constraint Graphs*, in 35th International Symposium on Algorithms and Computation (ISAAC 2024), J. Mestre and A. Wirth, eds., vol. 322 of LIPIcs, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024, pp. 51:1–51:14, https://doi.org/10.4230/LIPICS.ISAAC.2024.51.

[14] R. Impagliazzo and R. Paturi, *On the Complexity of k-SAT*, Journal of Computer and System Sciences, 62 (2001), pp. 367–375.

[15] G. D. Ita, P. Bello, and M. Rodríguez, *The Computation of #2SAT by a Fixed-Parameter Tractable Algorithm*, in Proceedings of the Eleventh Latin American Workshop on Logic/Languages, Algorithms and New Methods of Reasoning, Puebla, Mexico, November 15, 2018, 2018, pp. 101–113.

[16] G. D. Ita, J. R. Marcial-Romero, and J. A. H. Servín, *A Bottom-Up Algorithm for Solving #2SAT*, Log. J. IGPL, 28 (2020), pp. 1130–1140, https://doi.org/10.1093/JIGPAL/JZAA009.

[17] T. Laakkonen, K. Meichanetzidis, and J. van de Wetering, *A Graphical #SAT Algorithm for Formulae with Small Clause Density*, in Proceedings of the 21st International Conference on Quantum Physics and Logic, QPL 2024, Buenos Aires, Argentina, July 15-19, 2024, 2024, pp. 137–161, https://doi.org/10.4204/EPTCS.406.7.

[18] G. D. I. Luna, F. Z. Flores, and A. Rangel-Huerta, *A Note for Parametric Complexity of #2SAT*, in Proceedings of the Seventh Latin American Workshop on Non-Monotonic Reasoning, LANMR 2011, Toluca, Estado de México, México, November 7-8, 2011, 2011, pp. 95–104, https://ceur-ws.org/Vol-804/09_LANMR11.pdf.

[19] G. D. I. Luna and J. R. Marcial-Romero, *Computing #2SAT and #2UNSAT by Binary Patterns*, in Pattern Recognition - 4th Mexican Conference, MCPR 2012, Huatulco, Mexico, June 27-30, 2012. Proceedings, 2012, pp. 273–282, https://doi.org/10.1007/978-3-642-31149-9_28.

[20] M. A. L. Medina, J. R. Marcial-Romero, G. D. I. Luna, and J. A. H. Servín, *A Linear Time Algorithm for Counting #2SAT on Series-Parallel Formulas*, in Advances in Soft Computing - 19th Mexican International Conference on Artificial Intelligence, MICAI 2020, Mexico City, Mexico, October 12-17, 2020, Proceedings, Part I, 2020, pp. 437–447, https://doi.org/10.1007/978-3-030-60884-2_33.

[21] M. A. L. Medina, J. R. Marcial-Romero, J. A. H. Servín, and G. D. Ita, *Model Counting for #2SAT Problem in Outerplanar Graphs*, in Proceedings of the Eleventh Latin American Workshop on Logic/Languages, Algorithms and New Methods of Reasoning, Puebla, Mexico, November 15, 2018, 2018, pp. 76–87.

[22] J. Mei, M. M. Bonsangue, and A. Laarman, *Simulating Quantum Circuits by Model Counting*, in Computer Aided Verification - 36th International Conference, CAV 2024, Montreal, QC, Canada, July 24-27, 2024, Proceedings, Part III, 2024, pp. 555–578, https://doi.org/10.1007/978-3-031-65633-0_25.

[23] J. Mei, T. Coopmans, M. M. Bonsangue, and A. Laarman, *Equivalence Checking of Quantum Circuits by Model Counting*, in Automated Reasoning - 12th International Joint Conference, IJCAR 2024, Nancy, France, July 3-6, 2024, Proceedings, Part II, 2024, pp. 401–421, https://doi.org/10.1007/978-3-031-63501-4_21.

[24] A. Pilz, *Planar 3-SAT with a Clause/Variable Cycle*, Discret. Math. Theor. Comput. Sci., 21 (2019), https://doi.org/10.23638/DMTCS-21-3-18.

[25] M. Samer and S. Szeider, *Algorithms for Propositional Model Counting*, J. Discrete Algorithms, 8 (2010), pp. 50–64, https://doi.org/10.1016/j.jda.2009.06.002.

[26] F. Slivovsky and S. Szeider, *A Faster Algorithm for Propositional Model Counting Parameterized by Incidence Treewidth*, in 23rd International Conference on the Theory and Applications of Satisfiability Testing (SAT 2020), L. Pulina and M. Seidl, eds., vol. 12178 of Lecture Notes in Computer Science, Springer, 2020, pp. 267–276, https://doi.org/10.1007/978-3-030-51825-7_19.

[27] A. Sly, *Computational Transition at the Uniqueness Threshold*, in 2010 IEEE 51st Annual Symposium on Foundations of Computer Science, IEEE, 2010, pp. 287–296.

[28] L. G. Valiant, *The Complexity of Computing the Permanent*, Theor. Comput. Sci., 8 (1979), pp. 189–201, https://doi.org/10.1016/0304-3975(79)90044-6.

[29] L. G. Valiant, *The Complexity of Enumeration and Reliability Problems*, SIAM J. Comput., 8 (1979), pp. 410–421, https://doi.org/10.1137/0208032.

[30] D. Zuckerman, *On Unapproximable Versions of NP-Complete Problems*, SIAM J. Comput., 25 (1996), pp. 1293–1304, https://doi.org/10.1137/S0097539794266407.